ICS 451: Today's plan

• Sockets API, continued
  – C
  – Windows
• C programming reminders
Socket API in C: server calls

- **listen** makes a socket a server socket
  
  ```c
  int listen(int sockfd, int queue);
  ```

  - Use a small value such as 5 for the queue size

- **Bind** selects a local port
  
  - do check that the return value is 0

  ```c
  sin.sin_family = AF_INET;
  sin.sin_port = htons (server_port_number);
  sin.sin_addr = INADDR_ANY;

  if (bind(sockfd, (struct sockaddr *) (&sin), sizeof (sin)) != 0) perror ("bind")
  ```
Socket API in C: binding with IPv6

```c
struct sockaddr_in6 sin6;

sin6.sin6_family = AF_INET6;

sin6.sin6_port = htons (server_port_number);

sin6.sin6_addr = in6addr_any;

if (bind(sockfd, (struct sockaddr *) (&sin6),
         sizeof (sin6)) != 0) perror ("bind v6");
```
Socket API in C: server accept

- **accept** creates new sockets from server socket
  
  ```c
  int accept(int sockfd, struct sockaddr * peer,
              socklen_t * addrlen);
  ```

- The return value (if >= 0) can be used with **send** and **recv**

- The memory that peer points to is filled with the peer's address, up to *addrlen bytes

- Then *addrlen is set to the size of the address
Socket API in C: close

- **close** closes a server or client socket
  
  ```
  int close (int fd);
  ```

- **shutdown** can be used to declare that we will stop reading from or writing to a socket
  
  ```
  int shutdown (int sockfd, int how);
  ```

- how is **SHUT_RD** or **SHUT_WR** (**SHUT_RDWR** is equivalent to **close**)

  ```
  ```
Socket API in C: send/recv

- **send** sends a buffer on a connected socket

  ```c
  int send (int fd, char * buffer, int len, int f);
  ```
  - Returns the number of bytes sent
  - Flags (f) will normally be 0

- **recv** receives data from a connected socket

  ```c
  int recv (int fd, char * buffer, int len, int f);
  ```
  - Returns the number of bytes received (<= len)
    - or 0 if the socket was closed
  - Flags (f) will normally be 0
Socket API in C: sendto/recvfrom

- Used on unconnected (UDP) sockets
  ```c
  int sendto (int fd, char* buffer, int len, int f,
              struct sockaddr* to, socklen_t alen);
  ```
  - Similar to `send`, but takes address

- `recv` receives data from a connected socket
  ```c
  int recvfrom(int fd, char* buffer, int len, int f,
               struct sockaddr* a, socklen_t* alen);
  ```
  - Address and alen filled in as for accept
Sockets API on Windows

• Windows requires a call to `WSAStartup` prior to using the sockets API
  - Parameters are the version of the latest Winsock DLL that is acceptable, and a data structure to be filled with information about the implementation in use
• Must use `closesocket` instead of `close`
• May call `WSACleanup` after finishing
• May have to `#include <sys/socket.h>`
C programming reminders

- Character arrays have a fixed size, cannot grow or shrink
- Pointers have to point somewhere
  - Where are your pointers pointing?
- Use strncpy (or snprintf) in preference to strcpy
- Strings are null-terminated
  - Data received from the network usually is not
- A null character is different from a null pointer!
Exercise: spot the bugs

char * a = NULL;
*a = 'x';
char b [1000];
a = b + 10;
strncpy (a, "foo", sizeof (a));
int n = recv (s, a, strlen (a), 0);
a [n] = '\0';
return a;
Exercise: write C code

• Declare two char arrays, str1 of size 555 and str2 of size 10
  - You should use constants for the array sizes
  - Be sure that you declare the constants!

• Receive a string from socket sock into str1
  - sock is already connected

• Make str1 into a string and print it

• Copy the first part of str1 into str2

• Make str2 into a string